Code the Globe: Interactive Content for Spherical Displays with simple Webpages

Thomas Crespel, Brett Ridel, Clara Rigaud, Anke Brock, Patrick Reuter

Code the Globe: Interactive Content for Spherical Multi-Touch Displays with Simple Webpages

Thomas Crespel
Inria Bordeaux, France
thomas.crespel@inria.fr

Brett Ridel
Inria Bordeaux, France
brett.ridel@inria.fr

Clara Rigaud
University Bordeaux, France
clara.rigaud@u-bordeaux.fr

Anke M. Brock
Inria, LaBRI, Univ. Bordeaux
anke.brock@inria.fr

Patrick Reuter
Inria, LaBRI, Univ. Bordeaux
preuter@labri.fr

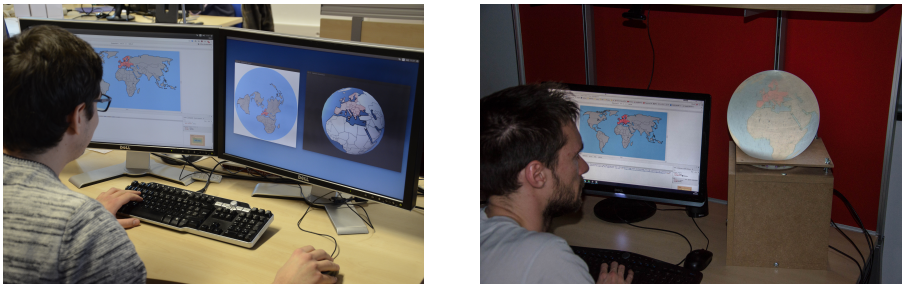

Figure 1. (left) Prototyping a GUI application with a 3D preview on the desktop, and (right) deployment on an interactive spherical display.

ABSTRACT

Although spherical displays become more and more pervasive, the design of interactive content for these displays still remains difficult as it requires most developers to get familiar with specific tools for managing the output and input. In this paper, we present a framework for developing applications for multitouch spherical displays that makes it possible to create interactive content by programming standard GUI applications, as for example interactive web pages. The principal idea is to adapt the window output and interaction input of classical GUIs outside the application. To this end, our framework consists of two standalone applications where the first one captures the window output and changes the projection via GPU shaders, and the second one adapts the input with a Node.js server and sends interaction and mouse events. In this way, the same application runs on a standard desktop, and on the spherical display. Advantages of our approach include fast prototyping, and the fact that masses of developers can create applications for spherical displays just as if it were, for example, classical web applications. We believe that our framework will contribute to making spherical displays even more pervasive in the future.

PerDis'17, June 07–09, 2017, Lugano, Switzerland

ACM ISBN 978-1-4503-5045-7/17/06…$15.00

DOI: http://dx.doi.org/10.1145/3078810.3078828

ACM Classification Keywords

I.3.3 Computer graphics: Display algorithms; I.3.6 Computer graphics: Interaction techniques; H.5.2 User Interfaces: Graphical user interfaces (GUI), Input devices and strategies (e.g., mouse, touchscreen)

Author Keywords

spherical display; touch input; web programming

INTRODUCTION

The presence and use of interactive displays in our everyday life is rapidly evolving. Only a decade ago, we mostly used desktop PCs with a mouse, and today, multitouch smartphones and tablets are our everyday compagnons. Beyond planar displays, recent OLED technologies made curved displays available, and the use of video projectors makes it even possible to project on and interact with arbitrary surfaces in spatial augmented reality.

Interactive spherical displays present a fascinating display shape [1]. The displays sold today remain still quite expensive, and are used mostly for dissemination in exhibitions. We believe that these displays will soon be available to a greater public, and there are already first approaches for more affordable, budget-friendly spherical displays [6].

Although the potential of applications for spherical displays is high, for dissemination and for collaborative pedagogical projects such as geographic games, one issue of these non-planar displays is that it is not straightforward to create interactive content. Indeed, the content creator has to care about

the involved projection distortion of the input and output. Although dedicated development environments exist, our aim is to open the design of interactive applications for spherical displays to a maximum of developers, with a very steep learning curve.

Applications for spherical multitouch displays resemble classical GUI applications with touch input, except that the surface is closed and non-planar. Our idea is to let the content creators feel that they design a classical GUI application, while being able to put their applications in operational mode on non-conventional displays. To this end, we developed a framework consisting of two standalone applications: the first one captures the window output and changes the projection via GPU shaders, and the second one adapts the interaction input with a Node.js server and sends interaction and mouse events to the application. Spherical displays have a closed surface, and one of the most common interaction task is to rotate the content. This rotation is managed by our framework without necessarily notifying the GUI application: the rotational shift is directly sent to the output application, so that the content creator does not have to care about it.

One of the advantages of our framework is that it is independent from the type of the GUI application. As an example, we choose web applications that run in a browser, as interacting with a spherical display shares lots of interaction tasks and techniques that we do when using our web browser. Today, web developers and designers can be met around almost every corner, and so the creation of content for spherical displays can be done as if it was the design of a simple web page wrapped to the spherical display. We thus take advantage of the steadily growing power, simplicity, and user-friendlyness of web applications, and allow the creators to focus on the content, and not on technical issues.

The paper is organized as follows. In Section 2, we review related work. In Section 3, we detail our framework that we divide into three layers, the application layer, and the input and output processing layers. In Section 4, we show the four involved steps to transform a classical GUI application into a spherical display application, at the example of a web-based geographical game. In Section 5, we conclude and show avenues for future work.

RELATED WORK

Spherical displays: overview and hardware

Spherical displays are displays in the form of a globe. Users can explore different perspectives of the displayed data by physically moving around the sphere. Applications for spherical displays may include the exploration of virtual globes [5, 16], venue maps [17], or even teleconferencing [10].

Back in 1995, MacDonald[1] has built a first prototype of a spherical display consisting of four projectors that project animated images of to a sphere from the outside, with the aim to better represent global earth phenomena. A more space-saving approach is to rear project images on a translucent globe. To cover the entire globe, multiple projectors can be used [12].

[1]https://sos.noaa.gov/

With a single projector, one can either put a hemispherical convex mirror on the opposite side [8], or use a wide angle lens such as first done by Global Imagination [14]. For the latter, an azimuthal transformation must be applied to any image in order to achieve correct projection. We distinguish these spherical displays with projection from «crystal balls» where the content seems to float within a sphere [3, 7, 13], and displays composed of a number of LEDs [9].

All the formerly cited displays are not touch enabled, and the interaction is provided by additional devices such as tablets. But of course, direct multitouch interaction is the most efficient way to interact with content on the sphere, and the seminal research work on spherical multitouch displays can be attributed to Benko et al. [1]. They modified the Global Imagination product to make it interactive by touch with optical tracking: an infrared camera images the surface of the sphere from the inside, through the same lens as the projector, thus recording in azimuthal space. This raw image must be processed and transformed to achieve correct interaction with the content.

To our knowledge, the only commercial product that has the multitouch capability is sold by Pufferfish[2]. As major drawbacks, these products are quite expensive, and they do not provide a generic framework to easily develop applications for any interactive spherical display.

Yet, efforts have been made to produce low-cost spherical displays. For instance, Bolton et al. [2] constructed a DIY interactive spherical display. However, they did not provide the software for the correction of the projection distortion. Extending work from Patel [11], we recently [6] proposed a DIY approach to make an affordable spherical multitouch display, thus potentially generalizing the use of these interactive displays.

Spherical displays: software

For non touch-enabled displays, the software is generally limited to play back recorded, pre-rendered media such as images or videos. Interaction is done by additional devices such as tablets, often limited to basic control tasks such as selecting content or rewinding videos.

Touch-enabled spherical displays involve a wide angle lens, and in order to create interactive applications, the correct projection distortion has to be computed in real-time, both for input and output. In the Sphere project [1], the software was written in C# using Microsoft's XNA 2.0 framework with

[2]www.pufferfishdisplays.co.uk

Figure 2. Traditional representations of geographical data: a planar world map desk pad, and the spherical globe.

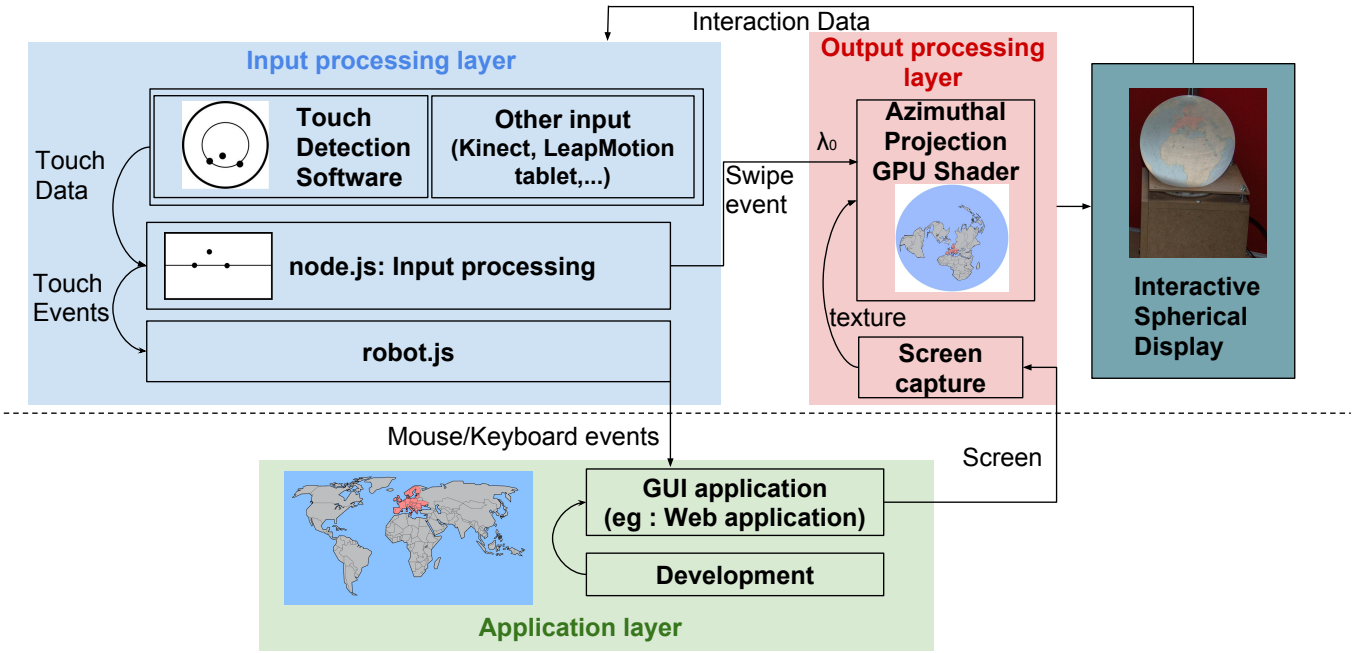

Figure 3. Our framework with the input and output processing layer on top of the application layer.

a custom vertex shader to handle the projection distortions. For the Globe4D project [5], the software was written in C++ and uses OpenGL for handling user input and rendering 3D animations supplied by the data layer. This data layer that provides textures for animating the sphere was written in C++ and Processing. In [6], the software was also implemented in C++ and OpenGL, and it also uses GPU shaders. The commercial touch-enabled spherical display from Pufferfish includes PufferWarp, a proprietary software that turns a video stream from cylindrical equidistant to azimuthal projection to fit on their product.

For creating interactive applications beyond video and for spherical displays in general, the prior projects require advanced programming skills for handling the projection distortions for input and output. We address this gap in the present work and free the content creators from these projection issues. We also want to emphasize that our work is compatible with GUI applications of any kind as long as they listen to mouse and keyboard events.

OUR FRAMEWORK

Overview

Our major idea to bring content development for interactive spherical displays to the masses is to let the content creator in his/her familiar environments. We are all used to the earth's representation like on desk pad world maps that we know from our childhood (Figure 2 (left)), which is a planar projection of our beloved classical globe (Figure 2 (right)). This planar projection of the earth is actually a parametrization of the 2D surface that is embedded in 3D space, with the *cylindrical equidistant projection* being one of the most popular. However, when plugging a spherical display to your computer as a secondary screen, this representation cannot be used since most of today's spherical displays are made of a projector in combination with a wide angle lens [1]: these projectors require a calibrated *azimuthal projection* (see the inset figure on the right). Similarly, the optical touch sensing of tactile spherical displays provides finger coordinates in the azimuthal projection.

For developing applications, we thus need to transform the output and input. For a calibrated spherical display, this transformation can be done via a 2D to 2D mapping, that converts the coordinates from the cylindrical equidistant projection to the azimuthal projection, or the other way around. This mapping must include the parameters from the calibration.

In the following subsections, we describe our framework that consists of three different layers: an application layer, as well as an input processing and output processing layer that makes the application compatible with a deployment on an interactive spherical display. For an illustration, consider Figure 3.

Application Layer

The application layer consists in a classical, interactive GUI application. The graphical output of the GUI application has a specific area in cylindrical equidistant projection to be displayed on the sphere, and possibly additional areas for information and control. We decided to use a web application as the GUI, since the development is fast and has a broad designer and developer community. Typical useful interaction widgets to be shown on a sphere are buttons for launching

actions, sliders for forwarding or rewinding videos, and also draggable elements. Think of, for example, a pedagogical geography game where the user has to select a continent from its name (as here Europe in the example of Figure 3), or to drag a country flag on the corresponding country. In the case of web applications, lots of handy tools exist, such as JQuery UI for making elements draggable or adding fade effects with a simple line of code. Note that the GUI application is the same as for a desktop, and it can also be used independently from our framework.

Input processing layer

The input processing layer consists in a standalone application written in Node.js that processes touch input from the sphere. It can also process any other input that one would like to use to control the spherical display. As an example, a swipe in the air in front of a kinect or a leap motion sensor could be used to rotate the sphere.

Basically, the first gesture that almost every user tries when first seeing an interactive spherical display is a horizontal swipe touch gesture. This touch gesture is detected in this layer, and the rotational shift λ_0 corresponding to the longitude offset is modified. This rotational shift is then sent to the output processing layer. Note that the application layer is not necessarily notified about this change since the cylindrical equidistant projection remains in the same configuration, independent from the value of λ_0.

In our framework, we include only horizontal swipe gestures. This was initially inspired by the aim to imitate the use of classical globes for geographic applications where the north pole always remains on top, and to minimize the risk that users get lost. This being said, it would certainly be interesting to set up a user study to compare the effectiveness of introducing a second rotational shift ϕ_0, both for geographic applications and other types of applications that our framework supports as well.

For other touch input than swiping for rotation of the entire sphere, the touch coordinates have to be transformed from the azimuthal projection to the cylindrical equidistant projection. Then, they are sent to the application layer via mouse events. For simulating multitouch events, such as a pinch-and-zoom or rotate gesture, additional keyboard or mouse scroll wheel events can be sent that are then interpreted by the application. For sending the mouse and keyboard events, we use the Robot.js[3] desktop automation library.

Concerning the pinch-and-zoom gesture, we suggest to present zoom using an "overview and detail" representation [4] as it would be done with a magnifying glass in the real world. In this way, the global shape of the spherical display still corresponds to reality.

Output processing layer

The output processing layer consists in a standalone application written in C++/Qt that transforms the specific area of the output of the GUI application in the application layer from the cylindrical equidistant projection to the azimuthal projection. To this end, we capture the specific area of the output of the GUI application every frame in real-time, and use the screen grabbed image as a texture for transforming it into the azimuthal projection on the GPU with vertex and fragment shaders. The GPU has as input parameter the rotational shift λ_0 from the input processing, and so the spherical display takes into account the rotation.

[3] http://robotjs.io/

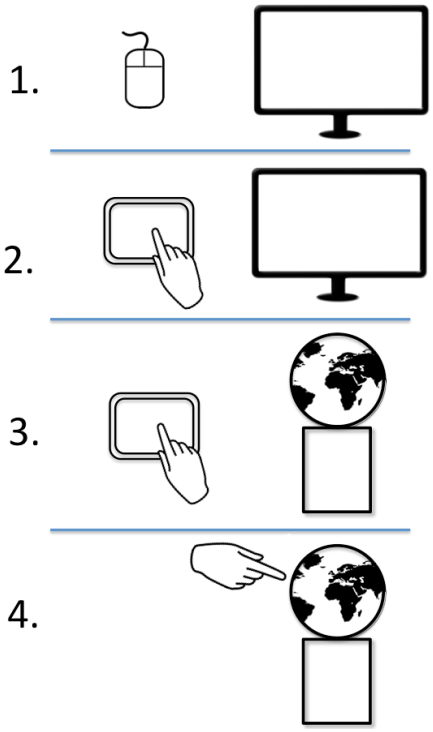

Figure 4. The 4 steps of the development process.

AN EXAMPLE APPLICATION WITH OUR FRAMEWORK

In this section, we detail the development process for creating and testing a concrete interactive application with our framework. For the display, we use the spherical touch display of Crespel et al. [6] (see Figure 3 on the right), and the application is a very simple geographical game where the users have to find continents on the globe.

For an illustration, consider Figure 4 that shows the 4 steps to follow for gradually transforming a classical GUI application into an interactive spherical display application, as detailed below.

In Step 1, the content creators start with the application layer and design a traditional 2D GUI application, such as a web application, based on classical mouse and keyboard input for defining the interactive elements. In our example, the area to project to the spherical display is a world map as an HTML image tag with HTML area map tags indicating the coordinates of the continents, as shown in Figure 1. The application logic is written in Javascript, where the user is asked to select a continent and rewarded with points for clicking on the right area. This step has the advantage that

the application can be designed without any other necessary hardware.

In Step 2, the content creators can enrich the input. To this end, the standalone application of the input processing layer has to be launched. Then, the creators can test touch input from a tablet, or other control from a smartphone, a kinect, or leap motion. This input is processed in the Node.js application, and depending on the touch input, either the appropriate mouse and key events are fired via Robot.js, or the rotational shift λ_0 is updated. Note that at this step, there is no transformation between projections since the input is planar and thus coherent to the cylindrical equidistant projection. In our example, we take control of the mouse cursor on the tablet that emits the same protocol for touch events as used in the spherical display, namely the TUIO protocol. The users can select continents on the tablet (distant interaction) by a tap gesture that is mapped to a *mousemove* event, and validate by a one-second press gesture that is mapped to a *mouseclick* event. This step has the advantage that it tests the input processing layer with a simple tablet and without requiring a spherical display.

In Step 3, the content creators can deploy the output to the spherical display, in addition to the output on the screen. To this end, the standalone application of the output processing layer has to be launched. This application captures the screen and takes into account the current rotational shift λ_0 in order to produce the correct projection transformation to send to the spherical display. In our example game, the spherical display is plugged as a secondary display, and in addition to selecting the continent by indirect distant touch interaction, the users can rotate the content of the spherical display with a swipe gesture on the tablet. This step has the advantage that it tests the output of the spherical display, for example whether it is correctly calibrated.

In the final Step 4, the touch input is taken from the spherical display, most often by means of optical tracking with a camera, and the input processing layer already used in Step 2 is extended by transforming the coordinates from the touch input to cylindrical equidistant projection in the Node.js standalone application. In our example game, we use the CCV interface[4] for the visual finger tracking that sends TUIO messages to the Node.js input processing application. The users can now interact with direct touch on the sphere for selecting continents, or for rotating the entire content. This step has the advantage that it tests the direct touch input of the spherical display.

Following these steps, most of the different critical issues when developing an interactive application for a spherical multitouch display can be addressed independently.

CONCLUSION & AND FUTURE WORK

We presented a framework for rapidly developing interactive content for spherical displays as if they were classical GUI applications. The framework also enables to test independently the different critical issues involved in spherical multitouch displays, such as calibration, touch events, or simply the application logic. Although we consider our method as a powerful tool, especially for prototyping, there are still some open problems that we consider to address in the future.

First, concerning the input processing that transforms multitouch events into mouse/keyboard events, the predominant touch cursor is mapped to mouse events, and of course there can only be one mouse cursor in the applications. So in reality, to simulate multi-touch events, we have to explicitly specify the mapping in the Node.js server that has then to be manipulated by the developer, what we wanted to avoid in the first place. We are currently investigating the simulation of touch events via websockets that can then be interpreted by the application.

Second, concerning the output processing, there is an issue with the cylindrical equidistant projection when the users want to drag elements around the corner, e.g. leaving on the right and entering on the left. As dragged elements are usually larger than the one pixel of the hotspot of the mouse cursor, they have to be rendered twice to be correctly projected.

Third, for the involved projection transformation in general, note also that all the interactive elements such as text or other boxes are distorted. In the future, we want to address this problem following Vega et al. [15] and deform these elements in the GPU shader in order to obtain correct output.

Despite these current issues, we are convinced that our framework is a step forward when it comes to fast prototyping of applications for interactive spherical displays. Remember also that although the initial motivation of this work was to rapidly prototype applications for web browsers, it can be extended for other GUI applications than web browsers.

We believe that this paper will contribute to making spherical displays more pervasive in the future by enabling graphic and web designers to easily develop applications. We are planning to exploit this potential, and, of course, develop a multitude of applications on the spherical display. Beyond geographical applications, we also think of further domains, as for example for teaching mathematical concepts that are better understandable on spherical displays. Moreover, we will focus on how people interact with these spherical displays.

ACKNOWLEDGEMENTS

This work was supported by the ISAR project ANR-14-CE24-0013.

REFERENCES

1. Hrvoje Benko, Andrew D. Wilson, and Ravin Balakrishnan. 2008. Sphere: multi-touch interactions on a spherical display. In *Proceedings of the 21st annual ACM symposium on User interface software and technology - UIST '08*. ACM Press, New York, New York, USA, 77. DOI:http://dx.doi.org/10.1145/1449715.1449729

2. John Bolton, Kibum Kim, and Roel Vertegaal. 2011. SnowGlobe: A Spherical Fish-tank VR Display. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems (CHI EA '11)*. ACM, New York, NY, USA,

[4]Community Core Vision, http://ccv.nuigroup.com

1159–1164. DOI: http://dx.doi.org/10.1145/1979742.1979719

3. Li-Wei Chan, Yi-Fan Chuang, Meng-Chieh Yu, Yi-Liu Chao, Ming-Sui Lee, Yi-Ping Hung, and Jane Hsu. 2007. Gesture-based Interaction for a Magic Crystal Ball. In *Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology (VRST '07)*. ACM, New York, NY, USA, 157–164. DOI: http://dx.doi.org/10.1145/1315184.1315214
4. Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. 2009. A Review of Overview+Detail, Zooming, and Focus+Context Interfaces. *ACM Comput. Surv.* 41, 1, Article 2 (Jan. 2009), 31 pages. DOI: http://dx.doi.org/10.1145/1456650.1456652
5. Rick Companje, Nico van Dijk, Hanco Hogenbirk, and Danića Mast. 2006. Globe4D: Time-traveling with an Interactive Four-dimensional Globe. In *Proceedings of the 14th ACM International Conference on Multimedia (MM '06)*. ACM, New York, NY, USA, 959–960. DOI: http://dx.doi.org/10.1145/1180639.1180850
6. Thomas Crespel, Patrick Reuter, and Xavier Granier. 2017. A low-cost multitouch spherical display: hardware and software design. In *Display Week 2017*. The Society of Information Display, Los Angeles, California, United States. https://hal.inria.fr/hal-01455523
7. Tovi Grossman and Ravin Balakrishnan. 2006. The Design and Evaluation of Selection Techniques for 3D Volumetric Displays. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06)*. ACM, New York, NY, USA, 3–12. DOI:http://dx.doi.org/10.1145/1166253.1166257
8. Thomas Ligon, Christopher Fedde, and Jonathan Lang. 2003. A Self-Contained Spherical Display System. In *ACM Siggraph 2003 Emerging Technologies*.
9. Tamotsu Machida. 2002. GEO-COSMOS: World's First Spherical Display. In *ACM SIGGRAPH 2002 Conference Abstracts and Applications (SIGGRAPH '02)*. ACM, New York, NY, USA, 189–189. DOI: http://dx.doi.org/10.1145/1242073.1242202
10. Ye Pan and Anthony Steed. 2012. Preserving gaze direction in teleconferencing using a camera array and a spherical display. In *2012 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*. IEEE, 1–4. DOI: http://dx.doi.org/10.1109/3DTV.2012.6365433
11. Nirav Patel. 2011. Snow Globe: Part One, Cheap DIY Spherical Projection. pufferfishdisplays.co.uk. (2011). (Accessed on 11/30/2016).
12. F. Teubl, C. S. Kurashima, M. C. Cabral, R. D. Lopes, J. C. Anacleto, M. K. Zuffo, and S. Fels. 2014. Spheree: An interactive perspective-corrected spherical 3D display. In *2014 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*. 1–4. DOI:http://dx.doi.org/10.1109/3DTV.2014.6874768
13. Keita Ushida, Hiroshi Harashima, and Jun Ishikawa. 2003. i-ball 2 : An Interaction Platform with a Crystal-Ball-Like Display for Multiple Users. *Icat* (2003).
14. S.W. Utt, P.C. Rubesin, and M.A. Foody. 2008. Display system having a three-dimensional convex display surface. (2008). https://www.google.com/patents/US7352340 US Patent 7,352,340.
15. Karla Vega, Eric Wernert, Patrick Beard, C Gniady, David Reagan, M Boyles, and Chris Eller. 2014. Visualization on spherical displays: Challenges and opportunities. In *Proceedings of the IEEE VIS Arts Program (VISAP)*. 108–16.
16. Julie R. Williamson, Daniel Sundén, and Jay Bradley. 2015. GlobalFestival: evaluating real world interaction on a spherical display. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '15*. ACM Press, New York, New York, USA, 1251–1261. DOI: http://dx.doi.org/10.1145/2750858.2807518
17. Julie R. Williamson, Daniel Sundén, and Keith Hamilton. 2016. The lay of the land: techniques for displaying discrete and continuous content on a spherical display. In *Proceedings of the 5th ACM International Symposium on Pervasive Displays - PerDis '16*. ACM Press, New York, New York, USA, 38–44. DOI: http://dx.doi.org/10.1145/2914920.2915005